计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2017.20170444
54(10):2321-2333,2017

MNOS:拟态网络操作系统设计与实现

王禛鹏　扈红超　程国振
(国家数字交换系统工程技术研究中心　郑州　450003)
(whuwzp@whu.edu.cn)

Design and Implementation of Mimic Network Operating System

Wang Zhenpeng, Hu Hongchao, and Cheng Guozhen
(*National Digital Switching System Engineering & Technological Research Center*, *Zhengzhou* 450003)

Abstract　As a mission-critical network component in software defined networking (SDN), SDN control plane is suffering from the vulnerabilities exploited to launch malicious attacks, such as malicious applications attack, modifying flow rule attack, and so on. In this paper, we design and implement mimic network operating system (MNOS), an active defense architecture based on mimic security defense to deal with it. In addition to the SDN data plane and control plane, a mimic plane is introduced between them to manage and dynamically schedule heterogeneous SDN controllers. First, MNOS dynamically selects m controllers to be active to provide network service in parallel according to a certain scheduling strategy, and then judges whether controllers are in benign conditions via comparing the m responses from the controllers, and decides a most trusted response to send to switches so that the minority of malicious controllers will be tolerated. Theoretical analysis and experimental results demonstrate that MNOS can reduce the successful attack probability and significantly improve network security, and these benefits come at only modest cost: the latency is only about 9.47% lower. And simulation results prove that the scheduling strategy and decision fusion method proposed can increase system diversity and the accuracy of decisions respectively, which will enhance the security performance further.

Key words　software defined networking (SDN); active defense; mimic security defense; dynamic heterogeneous redundancy; network operating system (NOS)

摘　要　控制层的漏洞利用攻击,如恶意 APP、流表篡改等是软件定义网络(software defined networking, SDN)面临的主要威胁之一,而传统基于漏洞修复技术的防御策略无法应对未知漏洞或后门.提出一种基于拟态防御思想的网络操作系统安全架构——拟态网络操作系统(mimic network operating system, MNOS)——保障 SDN 控制层安全.该架构采用异构冗余的网络操作系统(network operating system, NOS),并在传统的 SDN 数据层和控制层间增设了拟态层,实现动态调度功能.首先拟态层动态选取若干 NOS 作为激活态并行提供服务,然后根据各 NOS 的处理结果决定最终的有效响应返回底层交换机.实验评估表明:在增加有限的时延开销下,MNOS 可以有效降低 SDN 控制层被成功攻击的概率,

收稿日期:2017-06-11;**修回日期**:2017-08-01
基金项目:国家自然科学基金项目(61309020,61602509);国家自然科学基金创新群体项目(61521003);国家重点研发计划项目(2016YFB0800100,2016YFB0800101);河南省科技攻关项目(172102210615,172102210441)
This work was supported by the National Natural Science Foundation of China (61309020, 61602509), the Foundation for Innovative Research Groups of the National Natural Science Foundation of China (61521003), the National Key Research and Development Program of China (2016YFB0800100, 2016YFB0800101), and the Key Technologies Research and Development Program of Henan Province of China (172102210615, 172102210441).
通信作者:扈红超(13633833568@139.com)

并具备良好的容错/容侵能力;在此基础上,提出的选调策略和判决机制,可以有效提升系统的异构度和判决的准确性,进一步提升安全性能.

关键词 软件定义网络;主动防御;拟态安全防御;动态异构冗余;网络操作系统

中图法分类号 TP393

软件定义网络(software defined networking, SDN)将传统数据平面和控制平面解耦,在逻辑上实现了网络的集中控制,使得网络配置、网络服务部署等都在网络操作系统(network operating system, NOS, 也称 SDN 控制器)之上实现,有利于简化网络管理.然而,SDN 应用实际生产环境时仍面临严峻的安全问题.首先,SDN 控制器集中管理着其覆盖网络的所有视图信息,因此,极易成为攻击的首选目标,带来额外的安全风险.其次,SDN 的重要特性是开放性,主要体现在可编程协议以及网络操作系统的开源社区,而这种开放性极易导致未知的安全漏洞等问题,例如已经曝出的 OpenDaylight 的 NetDump 漏洞、XML eXternal Entity (XXE)漏洞①、ONOS 的 DoS 漏洞②,以及 SDN 中的 Know Your Enemy (KYE)攻击[1]等.最后,SDN 控制层呈现的单一、静态特性有利于攻击者对其进行探测和分析.作为 SDN 的核心组件,控制器掌握着整个网络的视图,将上层策略转译为数据平面的转发规则,一旦被攻击或劫持,可导致信息泄漏,甚至网络瘫痪.

一般地,针对 SDN 控制层的攻击主要出于 2 个目的:操纵网络流量或瘫痪网络.操纵网络流量的攻击,一般利用控制器的漏洞,如控制器缺少对上层 APP 权限管理的漏洞,导致带毒 APP 下发恶意流表规则[2-3],以及控制器对 LLDP(link layer discovery protocol)包缺少安全认证(或者如 Floodlight,虽然提供鉴别码认证,但是鉴别码一直保持不变)的漏洞造成的拓扑伪造攻击和中间人流表篡改攻击[4]等.瘫痪网络的攻击,如存在恶意管理员[5],使用一个简单的 exit 命令致使 Floodlight 宕机造成网络瘫痪,或者通过发送探测报文推断控制器类型、消息处理逻辑和负载状态等信息和发动 DoS 攻击等[6].总之,利用控制器的漏洞发动攻击是控制器面临的主要威胁之一[7].

为应对上述安全问题,已有大量研究给出了防御方法.有研究者提出了为控制器增设北向接口的分级调用[8]和南向接口的限制访问[9]以解决上述北向 APP 权限和南向探测等漏洞问题,然而这类方法具有侵入性,需修改控制器南北向接口(甚至内核),并且这种"打补丁式"的防御手段无法根本上解决漏洞、后门等问题.因此,有研究者从架构上提出了分布式控制器[10-13]和弹性控制平面[14-15],这些结构可以有效解决 SDN 控制层呈现的静态特性问题和单点失效造成的网络瘫痪等攻击问题,然而目前主流的分布式架构中控制器仍是同构的,同一漏洞[16-17]即可能导致所有实例陷入瘫痪,另外,同基于拜占庭容错(Byzantine fault tolerance, BFT)机制的控制平面一样[18-19],上述方法控制器间都不可避免地相互通信,增大了攻击表面,易导致攻击感染.

本文将邬江兴院士提出的拟态安全防御思想[20]引入到 SDN 控制层,提出了拟态网络操作系统 (mimic network operating system, MNOS),一种具有动态异构冗余特性[21]的 SDN 安全架构,该架构在数据层和控制层间增加拟态层,动态调度若干异构控制器提供网络服务,并对其并行输出进行一致性判决,从中选取最可信的结果.本文的主要贡献有 3 个方面:

1) 基于拟态防御思想,设计了拟态网络操作系统架构,并对 Floodlight,Ryu,ONOS 这 3 种主流开源 SDN 控制器进行开发,实现了原型验证机 POC (proof of concept).实验评估证明,在增加一定的时延开销下,MNOS 可以有效保障 SDN 控制层安全;

2) 提出了一种基于异构度增益的选调策略,通过增加选调前后系统的异构程度,进一步提升 MNOS 的安全性能;

3) 优化 MNOS 的判决机制,在考虑先验知识的情况下,提升判决的准确性.

1 相关工作

SDN 在提供便于管理的控制层的同时,也使其极易遭受攻击,目前,针对控制层安全防御的研究,可以分为 2 个方向.

① Security: Advisories, https://wiki.opendaylight.org/view/Security_Advisories#.5BImportant.5D_CVE-2014-5035_netconf:_XML_eXternal_Entity_.28XXE.29_vulnerability

② Denial-of-Service (DoS) due to exceptions in application packet processors, https://gerrit.onosproject.org/#/c/6137/

1.1 改进型方案

改进型安全控制器的防御思路是在现有开源控制器的基础上,改进已有的安全服务或开发部署新的安全模块.目前,已有诸多研究者对 NOX,Floodlight 等控制器进行了安全功能的拓展,如 FortNOX[22], SE-Floodlight[23]等.

FortNOX 架构是 Porras 等人[22]针对 NOX 控制器设计的一种安全内核,为其提供基于角色的数据源认证、状态表管理、流表规则冲突分析、流表规则超时回调等安全功能,提升 NOX 控制器在流规则冲突检测方面的安全性能.在 NOX 基础上,Porras 等人又对 Floodlight 进行了类似安全拓展,设计了其安全增强版本 SE-Floodlight[23],增加了应用程序证书管理模块、权限管理等新的安全模块.

通过附加安全机制或修补安全漏洞的方式,使得研究者和开发人员比较被动,很难在短时间内有效提升控制器的安全性,并且这些"打补丁式"的改进型也无法从根本上解决控制层的安全漏洞问题.

1.2 革新型方案

针对 1.1 节的问题,一些研究者提倡在 SDN 控制器的设计和开发之初,便将安全性作为其核心问题之一进行考虑,从而突破已有控制器在系统架构、编程语言和预留接口等方面的限制,开发出全新的、内嵌安全机制的 SDN 控制器.

Shin 等人[24]提出了 Rosemary 架构,为实现对上层 APP 的管理,将其所有应用程序运行在一个封闭的应用程序区内,实时监控各个应用程序的行为,并通过检查各个应用程序的签名信息判定其是否为合法应用程序.然而,由于 Rosemary 系统对应用程序的签名方式是基于角色的签名机制,并将整个应用程序作为一个整体对其进行签名,因而无法较好地对应用程序各个模块的访问权限进行细粒度控制.Ferguson 等人[25]设计了内嵌安全机制 PANE 用于解决 SDN 中不可信用户访问请求之间的冲突问题.此外,针对控制器的单点失效问题,有研究者提出了分布式控制器,如 HyperFlow[10],作为应用程序运行于安装 NOX 的控制器之上,采用物理分布但逻辑集中的设计,因此可以在保证良好可拓展性的同时实现网络的集中管控,并且被动同步网络状态视图,赋予各个控制器决策权以降低数据层和控制层的时延.FlowVisor[11]则是在转发平面与控制平面之间引入了一个软件切片层实现控制消息切分.还有如文献[18-19]提出了基于拜占庭容错机制的控制层安全架构,通过拜占庭的一致性协议,检查点协议以及视图更换协议,实现多控制器间网络状态视图的一致以及对控制层的容错/容侵功能.在经典拜占庭模型下,当系统 $3f+1$ 个控制器时,可以最多容忍 f 个错误;而改进的拜占庭,可以只需 $2f+1$ 个即可实现容忍[26].

除此之外,在相关的防御手段上,如由美国网络与信息技术研究与发展(networking and information technology research and development, NITRD)计划近年来提出的移动目标防御(moving target defense, MTD)模型[27],其主要通过构建动态冗余网络增加不确定性,从而增加攻击难度.不同于拟态防御,MTD 每个周期只选取 1 个执行体提供服务.

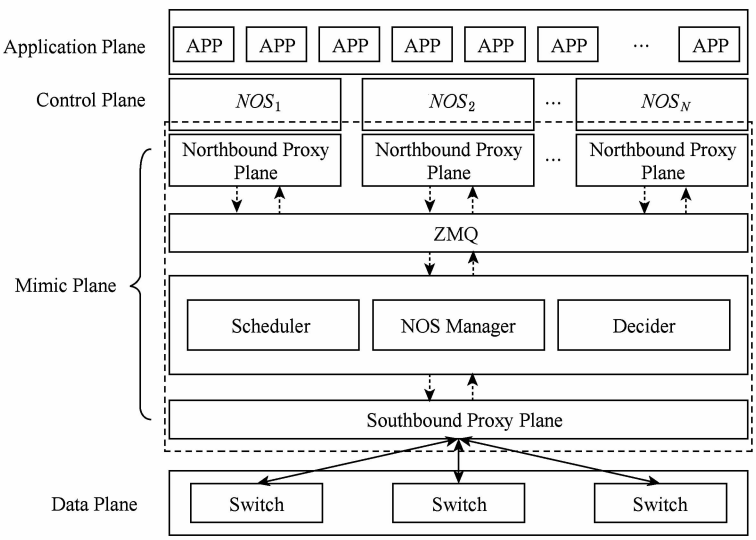

Fig. 1 The overview of MNOS

图 1 MNOS 整体架构

总的来说,上述分布式架构虽然可以有效单点失效等问题,但由于采用同构控制器,且控制器间相互通信,因此仍可能面临攻击感染等问题.

2 MNOS 架构

本文提出的拟态网络操作系统(MNOS)主要是将动态异构冗余模型引入 SDN 控制层中,以实现控制层内生安全性. 如图 1 所示为 MNOS 的整体框架,其中虚线框内即为本文提出的拟态层部分,下面对 MNOS 各模块分别进行介绍.

2.1 拟态控制协议

本文设计的拟态控制协议格式如图 2 所示:

0	1	2	3
version	type	nos_id/app_id	role
length			
xid			
data (OpenFlow)			

Fig. 2 Mimic Protocol

图 2 拟态控制协议

拟态控制协议运行在拟态层部分,即图 1 中虚线框内(虚线框内的虚线箭头表示拟态控制协议报文,框外的实线箭头表示 OpenFlow 协议报文). 之所以设计自定义协议而不直接采用现有的 OpenFlow 协议,是因为其不能满足本文的动态调度、判决(需要对 NOS 进行标识)等模块功能. 拟态控制协议类型(type)目前主要分为 2 类:1) 控制报文,主要是 NOS 与拟态层的交互,如 NOS 注册消息和保活消息;2) 数据报文,数据域 data 主要为 OpenFlow 等 SDN 南向协议报文. nos_id/app_id 字段主要是 NOS 及其上运行 APP 的标志,在 NOS 初始化注册时,由拟态层分配,便于拟态层管理. role 字段由拟态层标明 NOS 的角色,主要为 master 或 slave. xid 为会话标志,便于判决时比对.

2.2 南、北向代理层

南、北向代理层主要都是拟态控制协议的封装、解封装,即进入拟态层的报文封装为拟态控制协议,反之则解封装出数据域 OpenFlow 报文.

1) 南向代理层. 之所以增加南向代理层(实际上拟态核心层也可以完成其功能),主要是出于安全的考虑:拟态核心层功能复杂,涉及状态信息等,若直接对外呈现,则易受攻击(攻击面大). 而增加了功能实现较为简单的南向代理层后,从底层交换机的角度上看,南向代理层即为 SDN 控制器,从而实现拟态核心层对数据层的透明无感,从而保障 MNOS 的系统安全.

由上述分析,南向代理层可以采用专用硬件如 NetFPGA 实现,降低其遭受攻击的概率.

2) 北向代理层. 同南向代理层类似,北向代理层主要面向 SDN 控制器及其上运行的 APP 应用,因此从控制层的角度看,北向代理层被视为底层交换机.

由于本文采用异构的 SDN 控制器,如 Floodlight, Ryu, ONOS,具体实现不同,但实现方法和逻辑类似. 本文针对不同控制器类型开发 APP(模块),运行在 SDN 控制器上实现北向代理功能,其主要包含 2 个接口:与 SDN 控制器(以及 APP)间通信接口和与拟态核心层间通信接口. 逻辑功能主要为 SDN 控制器消息的拦截、报文封装和虚假交换机实现. 不同控制器"拦截"实现不同,如 Ryu 控制器,我们是通过重写 datapath 中的 send_msg 函数(增加封装、解封装拟态控制协议部分)实现的;而虚假交换机主要是为了完成底层交换机的"模拟",实现对控制器的透明无感.

2.3 消息队列

由于 MNOS 系统内部需要协调异构系统大量的通信,目前主流的异构系统通信的方法有 2 种:1) RPC(远程协议调用);2) 消息队列. 鉴于目前开源社区有较成熟的消息队列框架,本文选择基于消息队列的异构子系统通信实现.

就目前而言,主流的消息队列框架包括: RabbitMQ, ActiveMQ, ZeroMQ. RabbitMQ 是 AMQP 协议领先的一个实现,它实现了代理(Broker)架构,意味着消息在发送到客户端之前可以在中央节点上排队. 此特性使得 RabbitMQ 易于使用和部署,适宜于很多场景如路由、负载均衡或消息持久化等. 但是,这使得它的可扩展性差,速度较慢,因为中央节点增加了延迟,消息封装后也较大.

ZeroMQ (ZMQ) 是一个非常轻量级的消息系统,专门为高吞吐量/低延迟的场景开发,在金融界应用广泛,与 RabbitMQ 相比,ZMQ 支持许多高级消息场景. 更重要的是,由于本文需要在多个控制器间进行调度,同时涉及关闭某些控制器(进行清洗)后重新启动等操作,而 ZMQ 对连续的断开连接、重连接等都很友好,支持度高,适合本文应用场景,因此采用 ZMQ 实现.

2.4 拟态核心层

拟态核心层是 MNOS 功能核心部分,主要负责 NOS 的管理、调度和判决等功能,相应地,其逻辑架构包含 NOS 管理器、选调器和判决器.

2.4.1 NOS 管理器

NOS 管理器主要负责 NOS 管理和 NOS 状态维护 2 个功能,下面分别进行介绍:

1) NOS 状态维护. NOS 管理器维护的控制器状态信息如图 3 中 NOS 类所示:

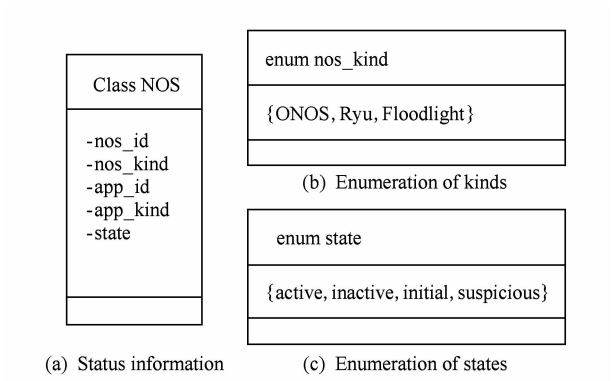

Fig. 3 Status information of NOS

图 3 NOS 的状态信息

每个新加入的控制器都必须在初始化阶段向 NOS 管理器注册,由管理器分配其唯一标识的 nos_id,并确定控制器类型 nos_kind,如 Floodlight,Ryu,ONOS 等;app_id 和 app_kind 同控制器注册时类似,这 2 个变量是为了维护该控制器上运行的 APP 信息;state 表示控制器的状态,主要有激活态(active)、空闲态(inactive)、初始态(initial)和可疑态(suspicious),状态 state 的变换过程如图 4 所示:

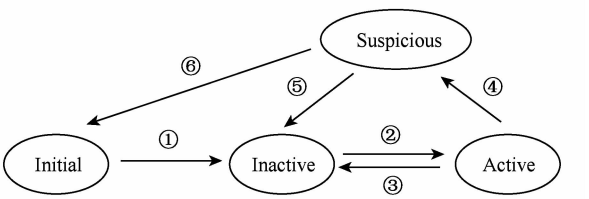

Fig. 4 State transforming of NOS

图 4 NOS 的状态变换

其中变换①由 NOS 管理器负责,主要工作是 NOS 初始化时的网络状态同步,由 NOS 管理器负责;变换②~③由选调器负责,确定控制器的工作状态、角色,这部分在 2.4.2 节介绍;变换④~⑥由判决器负责,主要发生在发现控制器可疑或者去可疑时,这部分在 2.4.3 节介绍.

2) NOS 管理. 得益于 docker、NFV、云等虚拟化技术,我们可以很方便地新增控制器(镜像)进行拓展,如图 4 所示,这里新增控制器也有可能是判决器发现可疑控制器,最后决定删除而重新初始化的控制器(变换⑥). 但新增控制器不能立即投入使用,因为涉及网络状态信息和 APP 状态等问题,因此首先需要对其进行网络状态同步,完成图 4 中变换①.

状态同步的步骤如图 5 所示:

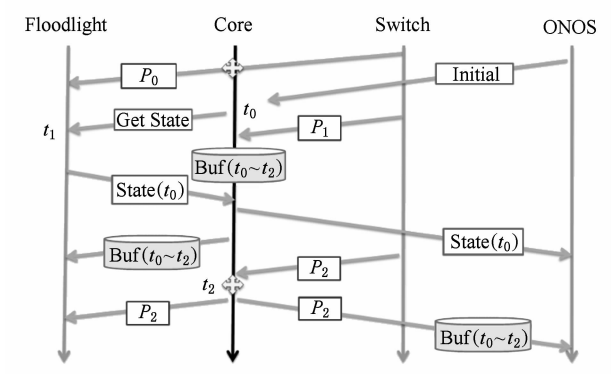

Fig. 5 State synchronization

图 5 状态同步流程

其中 Floodlight 表示正常运行的控制器,ONOS 为新增待同步的控制器,switch 表示底层交换机(实际应该分别为南、北向代理层,这里是为描述简便),core 表示拟态核心层. 以路由 APP 为例,具体步骤如下:

① 时刻 t_0 拟态核心层收到 ONOS 的初始注册消息,新建该控制器的状态信息,即 nos_id,nos_kind 等,并将状态设置为"初始态";

② 拟态核心层向 Floodlight 发送获取状态的拟态控制协议报文,同时将时刻 t_0 以后收到的 switch 报文进行缓存而不再发往 Floodlight(如图 5 中圆柱体);

③ 收到 Floodlight 发送的时刻 t_0 的状态(对于路由 APP,状态信息即为路由表,Floodlight 的节点路由表状态信息存放于 TopologyInstance 中的 pathcache 变量中)后,将其数据进行转换(适用于 ONOS 的数据)后发送给初始态的 ONOS 进行状态同步(此时同步的是时刻 t_0 的状态),同时将缓存的报文发往 Floodlight 进行处理;

④ 在时刻 t_2,Floodlight 正常处理报文,并将缓存的报文和后续到达的报文依次发往 ONOS 进行处理;

⑤ 在 ONOS 完成状态同步(处理完缓存报文后)向拟态核心层确认后,将其状态信息由"初始态"变换为"空闲态",等待选调模块的调用.

需要说明的是,若正常工作(激活态或空闲态)

的控制器中有相同类型的控制器,可直接用其进行同步,无需进行数据转换的步骤.

2.4.2 选调器

选调器主要负责控制器的调度,实现动态性,即图 3 中的变换②～③.为确保控制器的视图一致,本文采用类似 OpenFlow 协议的 slave/master 方法实现选调,不同的是 OpenFlow 协议中控制器集群只有一个 master, slave 只有在 master 无法正常工作时才修改为 master(主要是应对单点失效等故障),而本文是将多个控制器同时设为 master.具体地,若北向代理层收到的拟态控制协议报文中 role 字段为 master,则将其交付控制器处理并正常下发消息;反之,若为 slave,则北向代理交付控制器处理后对其消息进行拦截而不下发.

因此,首先选调器根据选调策略选取出 m(m 一般为不小于 3 的奇数)个控制器后,将其状态信息由空闲态设为激活态,其余设为空闲态,而后只需根据各控制器的状态,在分发消息给各控制器时,在 role 字段填入 slave/master (0/1) 即可:对于空闲态和初始态的控制器,该字段设为 slave(初始化的控制器不具备正常工作的能力);激活态和可疑态的,设为 master(由于可疑的控制器需要进一步考察,因此需要其回复所有的消息请求,便于判决器与其他控制器的响应进行比对,确认其可信与否).具体的选调策略将在第 3 节进行介绍.

2.4.3 判决器

判决器主要负责对多个控制器的响应报文进行判决,判定最可靠的响应.判决机制在数据融合等领域有很多研究,以最简单的大数判决为例.当判决器收到 m 份响应中(由 2.4.2 节可知,只有 role 字段为 master 的控制器才会响应)有大于或等于(m+1)/2 份结果一致时,则判定该结果为有效响应,发往南向代理层(最终到交换机);而对于其余和该结果不一致的控制器,则判定为可疑,将其状态由激活态改为可疑态(图 3 中的变换④),然后着重对其进行考察,若后期仍多次出现不一致的情况,则向 NOS 管理器通告,将其删除,并重新初始化新的控制器实例(图 3 中变换⑥),反之,则重新设为空闲态(去可疑,图 3 中变换⑤).

上述判决机制是基于一个假设:多数控制器同时被攻击且攻击后输出一致的错误响应的概率极低.尤其是本文采用的异构控制器,攻击者需要对多种控制器进行漏洞挖掘,因此攻击者成功实施攻击的难度将会进一步增加[16-17].具体的判决机制将在第 4 节进行介绍.

3 选调策略

第 2 节我们给出了选调器的工作原理,在本节中,我们详细介绍本文选调器的选调策略.

多样性,即异构配置,是指对集合中冗余实例采用相同功能但软件或硬件不同的配置,以增加系统的多样性[16-17].通过引入多样性增强网络安全的方法已成为研究热点并广泛应用,这种配置方法可以有效避免因同一种软件或硬件的漏洞、后门等造成共模故障,使整个冗余系统遭受毁灭性攻击[16-17].因此,本文提出一种基于异构度增益的选调策略,即最大化 MNOS 系统中控制器异构度.用到的符号、变量如表 1 所示:

Table 1 Definitions and Notations

表 1 变量及其含义

Notation	Definition
C	The Controller Set
C_{cur}	The Current Active Controller Set
C_{nxt}	The Next Active Controller Set
ϕ_c	The Threshold of Switching Cost
$m_{i,j}$	$m_{i,j}=\begin{cases}1, \text{Controller } i \text{ is switched to } j;\\ 0, \text{Others.}\end{cases}$
h_j^C	$h_j^C=\begin{cases}1, \text{Controller } i \text{ and } j \text{ are belonged to the same host;}\\ 0, \text{Others.}\end{cases}$
k_j^C	$k_j^C=\begin{cases}1, \text{Controller } i \text{ and } j \text{ are belonged to the same kind;}\\ 0, \text{Others.}\end{cases}$
DG	Diversity Gain
DG^{H}	Hardware Diversity Gain
DG^{K}	Software Diversity Gain

我们的选调目标是最大化系统选调前后的异构度增益 DG(由于每次选调的控制器数量受限,所以相当于激活态和空闲态控制器的动态迁移),异构度增益源于软硬件 2 个方面,即 DG^{H},DG^{K} 为最大化:

$$\sum_{C_{\text{cur}}\to C_{\text{nxt}}} DG = \sum_{C_{\text{cur}}\to C_{\text{nxt}}} (DG^{\text{H}} + DG^{\text{K}}). \tag{1}$$

DG^{H},DG^{K} 的计算方法如式(2)所示,且由于选调后切换的时延等因素,所以需对迁移开销(如迁移数量)进行限制:

$$\begin{aligned}&\text{s.t. } \forall j \in C\backslash C_{\text{cur}}, \sum_{i\in C_{\text{cur}}} m_{i,j} \leqslant 1,\\ &\sum_{i\in C_{\text{cur}}}\sum_{j\in C\backslash C_{\text{cur}}} m_{i,j} \leqslant \phi_c,\\ &DG^{\text{H}} = \begin{cases}\varepsilon_1, m_{i,j}=1 \wedge h_j^{C_{\text{cur}}}=1;\\ \varepsilon_2, m_{i,j}=1 \wedge h_j^{C_{\text{cur}}}=0.\end{cases}\\ &DG^{\text{K}} = \begin{cases}\partial_1, m_{i,j}=1 \wedge k_j^{C_{\text{cur}}}=1;\\ \partial_2, m_{i,j}=1 \wedge k_j^{C_{\text{cur}}}=0.\end{cases}\end{aligned} \tag{2}$$

显然,若新选取的控制器软件或硬件不同于当前激活态控制器集中的,则其带来的异构度增益大于相同的情况,因此,式(2)中变量关系满足 $\varepsilon_1<\varepsilon_2$, $\partial_1<\partial_2$.

式(2)最优模型是典型的 NP 难问题:当前激活态控制器在选调后变为空闲态,而空闲态被选取后变为激活态,相当于是一种映射算法.因此,本文给出如下启发式算法:每一轮选调都分为若干次迁移步骤,每一步都先从当前激活态集中选取出最破坏系统异构度的控制器,然后对其进行考察,按照式(1)(2)的原则选取候选控制器.最破坏系统异构度的控制器即某类型控制器数量最多的控制器.例如,某时刻激活态控制器集共有 4 种控制器,分别是 3 台 Ryu,一台 ONOS,一台 ODL 和 0 台 Floodlight,那么在进行这一轮选调时,我们优先选择其中的一台 Ryu 进行迁移(移出激活态集),并且由式(1)(2)可知,应当优先选择 Floodlight 加入候选控制器集中,以此类推.

算法 1. 选调算法.

输入:ϕ_c, C_{cur};

输出:C_{nxt}.

① $C_{\text{nxt}}=\phi$;

② while $\phi_c\neq 0$ and $C_{\text{cur}}\neq\varnothing$

③ 从C_{cur}中选择最破坏异构度的控制器 i;

④ for $j\in C\backslash C_{\text{cur}}$

⑤ if ($DG^{\text{H}}+DG^{\text{K}}$)最大

⑥ $C_{\text{cur}}\leftarrow j$;

⑦ end if

⑧ end for

⑨ $C_{\text{cur}}\leftarrow C_{\text{cur}}-i$, $\phi_c\leftarrow\phi_c-1$;

⑩ end while

⑪ if $C_{\text{cur}}=\varnothing$ and $\phi_c\neq 0$

⑫ 从$C\backslash C_{\text{cur}}$中选取($\phi_c-|C_{\text{nxt}}|$)个控制器至$C_{\text{nxt}}$中;

⑬ end if

⑭ $C_{\text{cur}}\leftarrow C_{\text{nxt}}$, $C_{\text{nxt}}\leftarrow\varnothing$;

⑮ return C_{cur}

假设控制器集共有$|C|$,则找到 C_{nxt}的时间复杂度可近似为 $O(\min(|C_{\text{cur}}|,\phi_c)(|C|-|C_{\text{cur}}|))$,所以复杂度最大为 $O(|C_{\text{cur}}|(|C|-|C_{\text{cur}}|))\leqslant O(|C|^2/4)$.

4 判决机制

判决器的任务是从各激活态控制器的响应报文中比对,抉择出最可信的响应回复给底层交换机,并记录可疑控制器.主要涉及 2 个关键点:比对内容和判决方法.在比对内容(粒度)方面,显然,若比对粒度为比特级,则计算量过大,因此,本文根据 OpenFlow 消息的格式进行字段级比对,着重考察 2 个关键字段:匹配(match)字段和行为(action)字段.如图 6 所示,攻击者通过劫持 SDN 控制器(NOS_1)篡改下发的流表规则,将原合法的转发逻辑 $match:in\_port=2$, $actions:output=1$,篡改为 $match:in\_port=2$, $actions:output=3$,企图引导用户流量到攻击者事先部署的 Web 服务器上.此时,判决器只需比对各控制器的 match 字段和 action 字段,从中选取较可靠的响应($output=1$)下发给交换机.

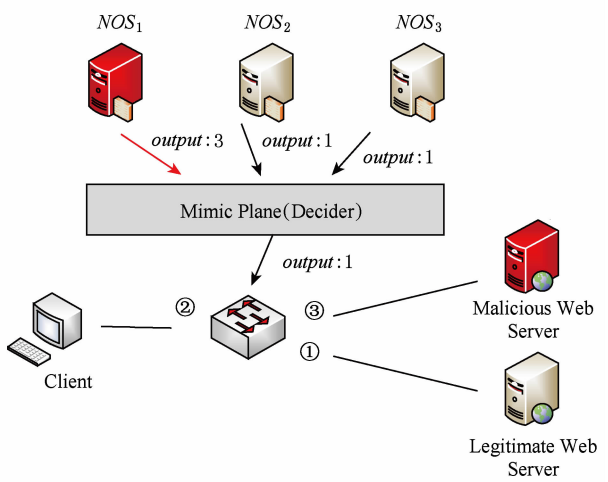

Fig. 6 Modifying flow rule attack

图 6 流表篡改攻击

上述示例采用的判决方法是 2.4.3 节中简单的大数裁决机制(majority-bases decision),实现容错/容侵功能.但是这种方法多次判决之间相互独立(每一次判决只和当前接收的响应有关),无法有效利用先验知识,因此,本文优化判决机制,在考虑先验知识的情况下提升判决的准确性.

4.1 问题描述

假设有 m 个激活态控制器,存在若干拜占庭(恶意)控制器,每个控制器对同一 OpenFlow 消息(如 packet in 报文)进行处理后返回响应报文,我们用 $f=(f_1,f_2,\cdots,f_l)'$ 表示真正正确的响应报文,共有 l 个字段,其中 f_i 为 0/1 位序列,表示待比对的第 i 字段(例如 actions 中 $output$ 字段),$f_{i,j}$ 为该字段的第 j 位($f_{i,j}=\{0,1\}$, $j=1,2,\cdots,h_i$), h_i 为第 i 字段所占位数.如图 7 所示,其中 $o_{i,j}^k$(original)表示第 k 个控制器处理后的原始响应报文中第 i 字段的

第 j 位,而 $r_{i,j}^k$(report)则表示为该控制器实际向判决器返回的报文. 显然,一般而言,正常情况下应当满足 $o_{i,j}^k=r_{i,j}^k$,但是若控制器被攻击,变为恶意控制器,那么可能就有 $o_{i,j}^k\neq r_{i,j}^k$,假设恶意控制器反转该结果的概率为 P_{mal}($P_{\text{mal}}=P(o_{i,j}\neq r_{i,j})$). 因此判决器的目标就是在这些可能含有被篡改的报文中判决出最真实可靠的报文.

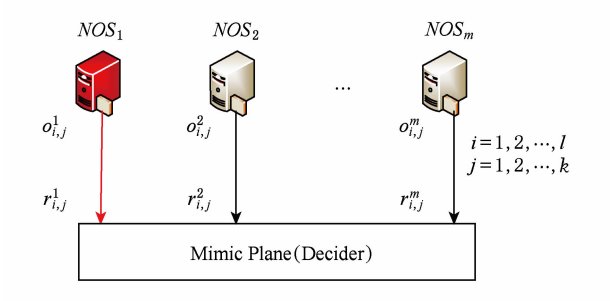

Fig. 7 Decision fusion scheme

图 7 判决场景

4.2 判决机制优化

由 4.1 节可知,判决器的任务可以转化为求解:

$$f^*=\arg\max_f P(f|r). \tag{3}$$

由贝叶斯定理和所有可能的响应都是等概率发生可知,式(3)等价于:

$$f^*=\arg\max_f P(r|f). \tag{4}$$

我们用 $a_n=(a^1,a^2,\cdots,a^m)$ 的 0/1 序列表示控制器集的状态(恶意或正常):若 $a^k=1$,则表示 NOS_k 为恶意控制器;反之,则为正常控制器. 而 $P(a_n)$则表示当前激活态控制器集状态为该序列的概率,代入式(4)可得:

$$f^*=\arg\max_f\sum_{a_n}P(r\mid a_n,f)P(a_n)=$$
$$\arg\max_f\sum_{a_n}\Big(\prod_{k=1}^{m}\prod_{i=1}^{l}\prod_{j=1}^{h_i}P(r_{i,j}^k\mid a^k,f_{i,j})\Big)P(a_n). \tag{5}$$

可以看出,由于乘法效应,式(5)求解较为复杂,我们采用类似 OpenFlow 多级流表的思想,将式(5)分解转化为多级相加,即单独求解各个字段的最优解,可得:

$$f_i^*=\arg\max_f\sum_{a_n}\Big(\prod_{k=1}^{m}\prod_{j=1}^{h_i}P(r_{i,j}^k\mid a^k,f_{i,j})\Big)P(a_n). \tag{6}$$

于是由各个字段的解可以组合出最优解 f^*. 此外,我们考虑存在系统误差概率 ε, $\varepsilon=p(o_{i,j}\neq f_{i,j})$ 用于描述控制器的系统错误概率,于是正常控制器和恶意控制器最后返回给判决器的结果不同于真正正确的结果的概率可以分别表示为

$$P(r_{i,j}^k\neq f_{i,j}\mid a^k=0)=P((o_{i,j}^k\neq f_{i,j}\wedge o_{i,j}^k=r_{i,j}^k)\mid a^k=0)=P(o_{i,j}^k\neq f_{i,j}\mid a^k=0)=\varepsilon, \tag{7}$$
$$P(r_{i,j}^k\neq f_{i,j}\mid a^k=1)=P((o_{i,j}^k\neq f_{i,j}\wedge o_{i,j}^k=r_{i,j}^k)\mid a^k=1)+P((o_{i,j}^k=f_{i,j}\wedge o_{i,j}^k\neq r_{i,j}^k)\mid a^k=1)=\varepsilon(1-P_{\text{mal}})+(1-\varepsilon)P_{\text{mal}}. \tag{8}$$

因此,将式(7)(8)的结论代入,可将式(6)转化为

$$f_i^*=\arg\max_f\sum_{a_n}\Big(\Big(\prod_{k;a^k=0}(1-\varepsilon)^{n_{eq}^k(i)}\varepsilon^{h_i-n_{eq}^k(i)}\Big)\times\Big(\prod_{k;a^k=1}(1-\delta)^{n_{eq}^k(i)}\delta^{h_i-n_{eq}^k(i)}\Big)\Big)P(a_n), \tag{9}$$

其中,$\delta=P(r_{i,j}^k\neq f_{i,j}\mid a^k=1)$(如式(8)所示),$n_{eq}^k(i)$ 表示第 k 个控制器返回的响应报文的第 i 个字段中与 f_i 中对应位相同的数量,即 $n_{eq}^k(i)=\sum_{j=1}^{h_i}(r_{i,j}^k==f_{i,j})$.

我们假设攻击者对每个控制器以相同的概率 α 成功实施攻击,使其变为恶意控制器,并且由于控制器间相互异构,所以可以认为各个控制器被攻击的状态相互独立,因此:

$$P(a_n)=\prod_{k=1}^{m}P_A(a^k), \tag{10}$$

其中,当 $a^k=1$ 时,$P_A(a^k)=\alpha$(一般假设 $\alpha<0.5$,否则,若半数以上为恶意控制器,则判决器就不可能做出正确的决策). 将式(10)代入式(9),可得:

$$f_i^*=\arg\max_f\prod_{k=1}^{m}((1-\alpha)(1-\varepsilon)^{n_{eq}^k(i)}\varepsilon^{h_i-n_{eq}^k(i)}+\alpha(1-\delta)^{n_{eq}^k(i)}\delta^{h_i-n_{eq}^k(i)}). \tag{11}$$

对于其他字段类似式(11)计算可得. 由式(11)可知,判决器一次完整数据融合最大的计算复杂度与激活态控制器个数 m 和比对的字段数 l 呈线性关系,而与每个字段所占位数 h_i 呈指数关系,因此 h_i 不宜过大,而 OpenFlow 协议 1.0 版本中,actions 中位数最大的为“modify Ethernet source/destination MAC address”(修改源、目的 MAC 地址)48 b,对此可以继续采用分解的方法,将其划为多个子字段,以降低每个子字段的位数.

上述演算的前提假设是判决器已知 P_{mal} 的值,然后由式(8)计算出 δ,代入式(11)求得最优解. 然而实际中,P_{mal} 的取值由攻击者确定,判决器本身并不可知,因此我们采用 2 人博弈论模型进行评估,即攻击者实际的 P_{mal}^A 和判决器采用的 P_{mal}^D,二者作为双方的博弈策略,而收益函数则采用判决错误率 P_e 进行衡量,同时将本文的判决机制同大数判决进行对比,实验结果在第 5 节介绍.

5 实验仿真

本节对 MNOS 性能、选调策略和判决机制进行实验评估. 我们采用 3 种开源控制器 Ryu, Floodlight, ONOS,在 9 台华为服务器(RH2288H V3)和 1 台 Pica8 (P3297)交换机上搭建实验环境,具体地,3 台运行 Floodlight、2 台运行 ONOS、4 台服务器运行 Ryu(其中 1 台同时运行南向代理层、1 台同时运行拟态核心层).

5.1 MNOS 安全性能

5.1.1 攻击成功概率仿真

首先对 MNOS 的安全性能进行评估对比. 本文主要与移动目标防御模型(MTD)[27]和基于拜占庭容错(BFT)的 SDN 控制层[18-19]架构进行对比.

由于 MTD 架构下,每个周期只选取 1 个执行体提供服务,因此,可认为 MTD 是拟态防御在 $m=1$ 时的特例. 并且由于改进的 BFT[26],可在 $2f+1$ 冗余情况下实现 f 个错误容忍,因此改进 BFT 在最终效果上等同于进行了大数判决,可视为静态 MNOS,即没有动态选调功能,只能一直由 m 个控制器提供服务.

在具体的仿真设置上,假设共有 N 个异构控制器,每个周期选取 m 个作为激活态,攻击者在一个周期的有效时间内可同时对 w 个控制器发起攻击(随机选取 w 个). 由于采用了异构配置,因此可以假设攻击不同控制器相互独立,且假设对每个控制器的攻击成功概率都为 P_a. 因此,MNOS 架构下若要攻击成功必须满足:1)有超过半数($(m+1)/2$)的激活态控制器被选择攻击;2)成功攻击. 因此攻击成功概率可表示为

$$P_{\mathrm{MNOS}}=\begin{cases}0, & w<(m+1)/2,\\ \sum\limits_{u=(m+1)/2}^{\min(w,m)}\dfrac{C_m^u C_{N-m}^{w-u}}{C_N^l}\sum\limits_{v=(m+1)/2}^{u}C_u^v P_a^v(1-P_a)^{u-v}, & \\ \text{others}, & \end{cases} \tag{12}$$

其中,前项积分表示攻击者选择攻击的 w 个控制器中有 u 个为激活态的概率,后项积分表示在 u 个激活态控制器中攻击成功 v 个概率和,这里采用大数判决机制,因此只有当 $u,v\geqslant(m+1)/2$ 才可能攻击成功.

而 MTD 架构下,攻击者选择正确的攻击目标的概率为 w/N,因此攻击成功概率可表示为(也可由式(12)取 $m=1$ 得到):

$$P_{\mathrm{MTD}}=\frac{w}{N}\times p_a. \tag{13}$$

而改进拜占庭架构下,攻击者需要成功攻击半数以上控制器,可表示为(也可直接由式(12)取 $N=m$ 得):

$$P_{\mathrm{BFT}}=\begin{cases}0, & w<(m+1)/2,\\ \sum\limits_{v=(m+1)/2}^{w}C_w^v P_a^v(1-P_a)^{w-v}, & \text{others}.\end{cases} \tag{14}$$

图 8 所示为攻击成功概率与 P_a 和每个时间间隔攻击者可同时攻击的控制器数量 w 的关系. 可见,整体上,攻击成功概率都随攻击能力 w 增加而增加;而 MTD 架构下,攻击成功概率呈线性增加,且远高于 MNOS 和 BFT 模型;而 BFT 模型也要高于本文的 MNOS 架构,由式(12)和式(14)对比可知,这是由于 MNOS 的动态选调降低了攻击者攻击激活态控制器的概率导致的,而当 $w\geqslant5$ 概率不再增加;而本文的 MNOS 安全性能明显优于其他 2 种架构(本文 MNOS 在 $m=5,w=11$ 时与 BFT 架构结果一致,因为此时攻击者可攻击全部控制器),且性能随 m 值的增加而增加,当然,时延开销等也会相应增加,后文将就时延代价等问题进行分析.

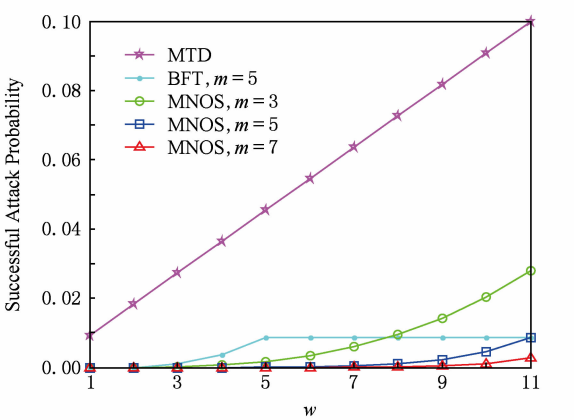

Fig. 8 Successful attack probability

图 8 攻击成功概率

5.1.2 入侵容忍能力验证

我们对 MNOS 的容错/容侵能力进行了测试,验证图 6 所示场景. 在拟态核心层判决器处收到 3 个控制器下发的 OpenFlow 的 FLOW_MOD(规则下发)报文如图 9 所示,我们修改了其中一个控制器上的 APP 代码(下发规则模块),模拟恶意 APP 攻击[28]. 可见,其中一个 *output* 字段由原 1 端口被篡改为 3 端口,此时判决器仍选择 1 端口(大数判决),实现容错/容侵. 显然,若激活态控制器数量为 m,则

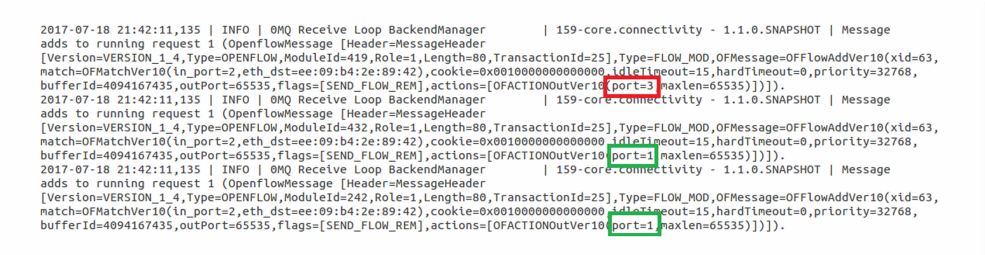

Fig. 9 Log file on decider

图 9 判决器 log 文件信息

若采用大数判决机制可容忍至多$(m-1)/2$个控制器故障,即攻击者至少需成功攻击$(m+1)/2$个控制器.并且由于采用了动态选调策略,限制了攻击的连续有效时间(一个切换周期),进一步增加了攻击难度.

5.1.3 时延开销及压力测试

最后由于引入了冗余控制器,并增设了拟态层,涉及选调、判决等模块,因此本文对传统单一控制器和 MNOS 的处理时延和吞吐量进行了对比分析评估.为了更好地描述和分析 MNOS 引入的额外时延开销,给出本节涉及的相关符号和定义,如表 2 所示:

Table 2 Definitions and Notations

表 2 变量及其含义

Notation	Definition
T_{td_ds}	The transmission delay between data plane and southbound proxy plane
T_{td_sm}	The transmission delay between southbound proxy plane and mimic core plane
T_{td_mn}	The transmission delay between mimic core plane and northbound proxy plane
T_{pd_s}	The processing delay of southbound proxy plane
T_{pd_m}	The processing delay of mimic core plane
T_{pd_n}	The processing delay of northbound proxy plane
T_{pd_c}	The processing delay of control plane
T_{add}	The added delay brought by MNOS

因此,传统单一控制器的时延可以表示为(这里的传输时延(含传播时延)为往返的时延):

$$T_{\text{original}}=T_{td\_ds}+T_{pd\_c}, \tag{15}$$

而 MNOS 的时延可以表示为

$$T_{\text{MNOS}}=T_{td\_ds}+T_{td\_sm}+T'_{td\_mn}+T_{pd\_s}+T_{pd\_m}+T_{pd\_n}+T_{pd\_c}, \tag{16}$$

其中,T'_{td_mn}表示 MNOS 架构下拟态核心层和北向代理层的往返传输时延,可以表示 $\max(T^1_{td\_mn}, T^2_{td\_mn},\cdots,T^m_{td\_mn})$($m$ 个控制器作为激活态),因为判决器需要等待所有报文进行判决,所以这部分时延由 m 个控制器中时延最大的决定,后文中简称之为 max 效应.

因此,额外增加的时延可以表示为

$$T_{\text{add}}=T_{td\_sm}+T'_{td\_mn}+T_{pd\_s}+T_{pd\_m}+T_{pd\_n}. \tag{17}$$

考虑到南、北向代理的工作主要为报文封装、解封装,这部分处理时延较小,将其忽略,则式(14)可近似为

$$T_{\text{add}}\approx T_{td\_sm}+T'_{td\_mn}+T_{pd\_m}. \tag{18}$$

因此,MNOS 架构引入的时延可以分为 2 部分:1)增加了多个逻辑层间的传输、传播时延 $T_{td\_sm}+T'_{td\_mn}$;2)拟态核心层的选调、判决(主要部分)处理时延 T_{pd_m}.

为分析上述各部分时延,下面给出 MNOS 的时延测试结果,我们采用图 5 所示的拓扑,测试 2 个主机间的 ping 包时延(删除了控制器中流表下发部分代码,目的是让所有数据包都经由控制器处理,便于测试并统计时延).结果如图 10 所示,其中 Original(加粗点划线)表示传统单一控制器.

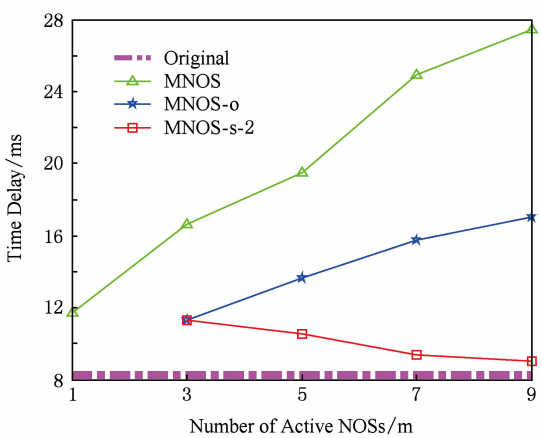

Fig. 10 Time delay test

图 10 时延测试

1) 传统单一控制器和无冗余 MNOS 对比.即图 9 中加粗点划线(Original)与三角划线的第 1 点(单个控制器即无冗余下的 MNOS)进行对比,可以

体现由 MNOS 新增的多逻辑层传输时延,即 $T_{td\_sm}+T_{td\_mn}$(由于只采用一个控制器,拟态层没有选调、判决等处理,因此 T_{pd_m} 忽略不计,并且无 max 效应),这部分时延增加约为 43.8%.

2) MNOS m 值不同的纵向对比. 即图 10 中三角划线,此时时延随控制器数量 m 增加而增加,与曲线第 1 点相比,这部分时延增加主要为 $T'_{td\_mn}+T_{pd\_m}$(T_{td_sm} 抵消),增加率分别约为 41.55%,66.14%,112.53%,134.17%.

可见 MNOS 时延随控制器数量增加较大,因此本文对 MNOS 进行优化,减少时延开销. 由上分析,结合式(18)可知,制约 MNOS 时延性能的主要因素为 T'_{td_mn} 和 T_{pd_m}(T_{td_sm} 本身无法避免),分别对应着 max 效应和拟态核心层的判决机制. 而相较于计算时延,传输、传播时延影响更大,因此本文从缓解 max 效应的角度对时延开销进行优化,提出预判决机制 MNOS-o,即当判决器收到 $(m+1)/2$ 个控制器的响应后先进行一次判决,若结果相同,则不必等待剩下 $(m-1)/2$ 个控制器的响应,结果如图 10 中五角划线所示,相比原方案,分别降低时延约 32.0%,30.0%,36.8%,37.91%. 进一步地,我们可以设置控制器安全阈值 q,例如取 $q=2$,只需 m 个响应中首先到的 2 个(而不是 $(m+1)/2$)一致,则下发其一致结果,而不等待剩余的,从而进一步降低时延,如图 10 中 MNOS-s-2 方块划线. 此时与传统单一控制器架构的时延增加最小仅约为 9.47%,可以推断,随着控制器数量的增加,时延将进一步降低,向其趋近(因为此时时延等于 $T^1_{td\_mn}, T^2_{td\_mn}, \cdots, T^m_{td\_mn}$ 中的第 q 顺序统计量,所以统计上看,m 越大,该值越小). 另外,预判决机制中,设置的安全阈值 q 越小,则时延越低,但相应地,系统的安全性能也就越低,因此可以先确定所需最低安全性能要求,然后参考图 8 数据确定合适的阈值 q.

然后我们采用 Cbench(controller benchmarker)对 MNOS 系统进行压力测试,测试不同交换机数量下的吞吐量,结果如图 11 所示.

加粗点划线为正常单个 Ryu 控制器架构的测试结果(Floodlight 和 ONOS 的吞吐量均为 $10^4\sim10^5$ 级,远大于 Ryu,因此 MNOS 性能主要受 Ryu 的影响). 可见,整体上吞吐量随交换机数量变化不大,而激活态控制器数量越多,则吞吐量越小. 同时,与单个 Ryu 的对比可见,MNOS 架构对网络吞吐量损害较大,同时延分析类似,这主要是由于判决机制导致的,性能仍有待提升.

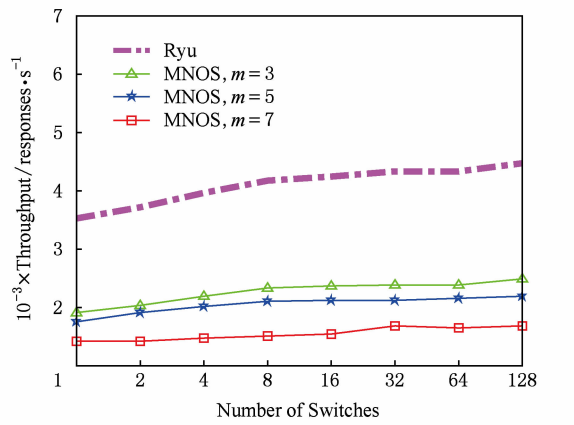

Fig. 11 Throughput test

图 11 吞吐量测试

5.2 选调策略评估

下面对选调策略进行仿真评估. 我们的参数设置为:共有 3 台主机、4 种控制器类型,每种有 3 个,ε_1, ∂_1 为 0.1, ε_2, ∂_2 为 0.3, 每轮选调开销 $\phi_c=3$. 仿真结果如图 12 所示,其中方形划线(Random1)为完全随机选调;圆圈划线(Random2)为考虑了异构度增益,但是随机选择待迁移的控制器,即算法 1 中第 3 步为随机选取而不是选择最破坏异构度的控制器优先迁移;三角划线(MNOS)即为本文提出的选调策略.

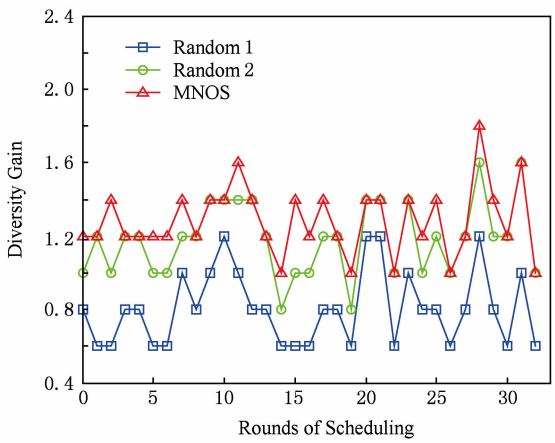

Fig. 12 Diversity gain

图 12 异构度增益

由仿真结果可以看出,本文提出的选调策略以及分步迁移算法可以增加激活态控制器集的异构程度,从而有效避免同构产生的攻击感染问题,增加攻击者攻击难度.

5.3 判决机制评估

下面对本文的判决机制进行仿真分析. 其中参数设置为:$m=21$, $h_i=4$(由 4.2 节分析,h_i 值不宜

过大,且字段所占位数都为 4 的倍数),$\varepsilon=0.1$,恶意控制器和判决器采用的 P_{mal} 取值分别为 0.5, 0.6,0.7,0.8,0.9,1.0. 我们进行 10 000 次重复试验,最终得到的博弈双方收益函数-错误率如表 3 所示:

Table 3 The Payoff (P_e)

表 3 收益函数(P_e)

P^A_{mal}	P^D_{mal}					
	0.5	0.6	0.7	0.8	0.9	1.0
0.5	0.0106	0.0106	0.0120	0.0131	0.0202	0.0243
0.6	0.0127	0.0144	0.0156	0.0186	0.0236	0.0260
0.7	0.0181	0.0201	0.0232	0.0258	0.0301	0.0352
0.8	0.0254	0.0310	0.0329	0.0398	0.0452	0.0455
0.9	0.0361	0.0419	0.0475	0.0550	0.0607	0.0647
1.0	0.0702	0.0744	0.0705	0.0698	0.0665	**0.0657**

表 3 中,当 $P^A_{\text{mal}}=P^D_{\text{mal}}=1.0$ 时,即为纳什均衡点(表 3 中加黑数字),此时当 $P^A_{\text{mal}}=1$ 保持不变且 $P^D_{\text{mal}}=1$ 时,错误率最低(判决器的最优策略);当 $P^D_{\text{mal}}=1$ 保持不变且 $P^D_{\text{mal}}=1$ 时,错误率最高(恶意控制器的最优策略). 我们在均衡点处,在完全相同参数下计算了采用大数判决机制的情况,此时错误率为 0.0862,本文机制可提升判决准确性约 31.2%. 可见,在博弈纳什均衡点时,本文方案可以提供更高的准确性,这是由于本文方案考虑了先验知识的情况(如恶意控制器所占比例、系统误差等),当然,其代价就是计算量(时延)的增加.

6 结束语

SDN 控制层作为软件定义网络中的核心管理模块,对全网的正常运行起着关键性作用,因此其安全性和可靠性需求不言而喻. 本文在分析已有的防御机制的基础上,基于拟态防御思想,提出了动态异构冗余模型的拟态网络操作系统 MNOS 架构. MNOS 通过构建异构冗余的网络操作系统,实现控制层的动态调度,增加控制层的不确定性,从而降低攻击成功概率. 为增加控制层的异构度,本文提出了基于异构度增益的选调策略,进一步增加攻击者的攻击成本. 最后,本文采用的判决机制,相比大数判决可以进一步提升判决的准确性.

当然,MNOS 架构也存在一定的不足,如实验仿真中代价分析部分所述,MNOS 由于采用冗余控制器共同决定最终结果,因此将会增加一定的时延,并对网络吞吐量损害较大. 对此,可以采用预判决机制进行改善. 后续的可能研究思路主要包括优化判决机制,降低计算复杂度;研究选调策略的优化,如根据 NOS 安全状态动态调整选调策略等.

参 考 文 献

[1] Conti M, Gaspari F D, Mancini L V. Know your enemy: Stealth configuration-information gathering in SDN [C] // Proc of the 12th Int Conf on Green, Pervasive, and Cloud Computing. Berlin: Springer, 2017: 386-401

[2] Scott-Hayward S, Natarajan S, Sezer S. A survey of security in software defined networks [J]. IEEE Communications Surveys & Tutorials, 2015, 18(1): 623-654

[3] Lee S, Yoon C, Shin S. The smaller, the shrewder: A simple malicious application can kill an entire SDN environment [C] //Proc of the 2016 ACM Int Workshop on Security in Software Defined Networks & Network Function Virtualization. New York: ACM, 2016: 23-28

[4] Hong Sungmin, Xu Lei, Wang Haopei, et al. Poisoning network visibility in software-defined networks: New attacks and countermeasures [C] //Proc of Network and Distributed System Security Symp. Reston, VA: ISOC, 2015: 8-11

[5] Matsumoto S, Hitz S, Perrig A. Fleet: Defending SDNs from malicious administrators [C] //Proc of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking. New York: ACM, 2014: 103-108

[6] Sonchack J, Aviv A J, Keller E. Timing SDN control planes to infer network configurations [C] //Proc of the 2016 ACM Int Workshop on Security in Software Defined Networks & Network Function Virtualization. New York: ACM, 2016: 19-22

[7] Lee S, Yoon C, Lee C, et al. DELTA: A security assessment framework for software-defined networks [C] // Proc of Network and Distributed System Security Symp 2017. Reston, VA: ISOC, 2017

[8] Lee C, Shin S. SHIELD: An automated framework for static analysis of SDN applications [C] //Proc of the ACM Int Workshop on Security in Software Defined Networks & Network Function Virtualization. New York: ACM, 2016: 29-34

[9] Wilczewski. Security considerations for equipment controllers and SDN [C] //Proc of IEEE Int Telecommunications Energy Conf. Piscataway, NJ: IEEE, 2016: 1-5

[10] Tootoonchian A, Ganjali Y. HyperFlow: A distributed control plane for OpenFlow [C] //Proc of the 2010 Internet Network Management Conf on Research on Enterprise Networking. Berkeley, CA: USENIX Associations, 2010

[11] Sherwood R, Gibb G, Yap K K, et al. FlowVisor: A network virtualization layer [EB/OL]. (2009-10-14) [2017-06-01]. http://archive. openflow. org/downloads/technicalreports/openflow-tr-2009-1-flowvisor. pdf

[12] Yeganeh S H, Ganjali Y. Kandoo: A framework for efficient and scalable offloading of control applications [C] //Proc of the 1st Workshop on Hot Topics in Software Defined Networks. New York: ACM, 2012: 19-24

[13] Koponen T, Casado M, Gude N, et al. Onix: A distributed control platform for large-scale production networks [C] //Proc of the 9th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Associations, 2010: 351-364

[14] Dixit A, Fang H, Mukherjee S, et al. Towards an elastic distributed SDN controller [C] //Proc of the 2nd Workshop on Hot Topics in Software Defined Networking. New York: ACM, 2013: 7-12

[15] Berde P, Hart J, et al. ONOS: Towards an open, distributed SDN OS [C] //Proc of the Workshop on Hot Topics in Software Defined Networking. New York: ACM, 2014: 1-6

[16] Voas J, Ghosh A, Charron F, et al. Reducing uncertainty about common-mode failures [C] //Proc of the 8th IEEE Symp Software Reliability Engineering. Piscataway, NJ: IEEE, 1997: 308-319

[17] Levitin G. Optimal structure of fault-tolerant software systems [J]. Reliability Engineering & System Safety, 2005, 89(3): 286-295

[18] Li He, Li Peng, Guo Song, et al. Byzantine-resilient secure software-defined networks with multiple controllers in cloud [J]. IEEE Trans on Cloud Computing, 2015, 2(4): 436-447

[19] Eldefrawy K, Kaczmarek T. Byzantine fault tolerant software-defined networking (SDN) controllers [C] //Proc of the 40th IEEE Computer Society Int Conf on Computers, Software & Applications. Piscataway, NJ: IEEE, 2016: 208-213

[20] Wu Jiangxing. Research on cyber mimic defense [J]. Journal of Cyber Security, 2016, 1(4): 1-10 (in Chinese)
(邬江兴. 网络空间拟态防御研究[J]. 信息安全学报, 2016, 1(4): 1-10)

[21] Hu Hongchao, Chen Fucai, Wang Zhenpeng. Performance evaluations on DHR for cyberspace mimic defense [J]. Journal of Cyber Security, 2016, 1(4): 40-51 (in Chinese)
(扈红超, 陈福才, 王禛鹏. 拟态防御 DHR 模型若干问题探讨和性能评估[J]. 信息安全学报, 2016, 1(4): 40-51)

[22] Porras P, Shin S, Yegneswaran V, et al. A security enforcement kernel for OpenFlow networks [C] //Proc of the 1st Workshop on Hot Topics in Software Defined Networks. New York: ACM, 2012: 121-126

[23] Porras P, Cheung S, Fong M, et al. Securing the software defined network control layer [C] //Proc of Network and Distributed System Security Symp 2010. Reston, VA: ISOC, 2010

[24] Shin S, Song Y, Lee T, et al. Rosemary: A robust, secure, and high-performance network operating system [C] //Proc of the 21st ACM Conf on Computer and Communications Security. New York: ACM, 2014: 78-89

[25] Ferguson A D, Guha A, Liang C, et al. Participatory networking: An API for application control of SDNs [C] //Proc of the ACM SIGCOMM 2013. New York: ACM, 2013: 327-338

[26] Veronese G S, Correia M, Bessani A N, et al. Efficient Byzantine Fault-Tolerance [J]. IEEE Trans on Computers, 2013, 62(1): 16-30

[27] Baker S. Trustworthy cyberspace: Strategic plan for the federal cybersecurity research and development program [EB/OL]. (2011-12) [2017-09-06]. https://www. nitrd. gov/SUBCOMMITTEE/csia/Fed _ Cybersecurity _ RD _ Strategic_Plan_2011. pdf

[28] Lee S, Yoon C, Shin S. The smaller, the shrewder: A simple malicious application can kill an entire SDN environment [C] //Proc of ACM Int Workshop on Security in Software Defined Networks & Network Function Virtualization. New York: ACM, 2016: 23-28

Wang Zhenpeng, born in 1993. Bachelor. His main research interests include software defined networking and mimic security defense.

Hu Hongchao, born in 1982. PhD and associate professor.. His main research interests include network security and software defined networking.

Cheng Guozhen, born in 1986. PhD and research assistant. His main research interests include software defined networking and active defense.